Program Obfuscation

Intuition: Mangle a program
• Same functionality as original
• Hides all implementation details

Potential uses:
• IP protection
• Prevent tampering
• Cryptography
Virtual Black Box Obfuscation [BGI+’01]

Having source code no better than black box access
Virtual Black Box Obfuscation

Potential Cryptographic Applications:

• Public key encryption from private key encryption:

  \[ \text{Enc} (\cdot) \rightarrow O \rightarrow P' \]

  \[
P( c_1 , c_2 , \oplus \in \{ + , \times \} ) \{ \\
  m_1 \leftarrow \text{Dec}(c_1) \\
  m_2 \leftarrow \text{Dec}(c_2) \\
  \text{return } \text{Enc}(m_1 \oplus m_2) \\
  \}
\]

• Homomorphic encryption:

• Functional Encryption
Virtual Black Box Obfuscation

Potential Cryptographic Applications:

• Public key encryption from symmetric key encryption:

• Homomorphic encryption:

  $$\text{Enc}(\cdot)$$

• Functional Encryption

  $$P'(c_1, c_2, \bigoplus \in \{+ , \times \}) \{$$

  $$m_1 \leftarrow \text{Dec}(c_1)$$

  $$m_2 \leftarrow \text{Dec}(c_2)$$

  return $$\text{Enc}(m_1 \bigoplus m_2)$$

  $$\}$$

Theorem ([BGI+’01]): VBB for all programs is impossible
Indistinguishability Obfuscation (iO) \([\text{BGI}^+01]\)

If two programs have the same functionality, obfuscations are indistinguishable

\[
P_1(x) = P_2(x) \quad \forall x
\]

\[
P_1' \approx P_2'
\]
Indistinguishability Obfuscation (iO)

BGI⁺ counter example does not apply to iO

An exploding field:
• [BGI⁺’01] Original definition
• [GR’07] Further investigation
• [GGH⁺’13] First candidate construction
  • Functional encryption
• [BR’13, BGK⁺’13, …] Additional constructions
• [SW’13, HSW’13, GGHR’13, BZ’13, …] Uses
  • Public key encryption, signatures, deniable encryption, multiparty key exchange, MPC, …
• [BCPR’13, MR’13, BCP’13, …] Further Investigation
Our Results

• Non-interactive multiparty key exchange without trusted setup
  • All existing protocols required trusted setup

• Efficient broadcast encryption
  • Distributed
  • Use existing keys

• Efficient traitor tracing
  • Shortest secret keys and ciphertexts known

All constructions from iO and one-way functions
(Non-Interactive) Multiparty Key Exchange

Public bulletin board

\[ K_{ABCD} \quad K_{ABCD} \quad K_{ABCD} \quad K_{ABCD} \]
Prior Constructions

First achieved using multilinear maps

• These constructions all require trusted setup **before** protocol is run
• Trusted authority can also learn group key
Prior Constructions

First achieved using multilinear maps

• These constructions all require trusted setup before protocol is run

• Trusted authority can also learn group key

params
Our Construction (w/ Trusted Setup)

Building blocks:

- iO
- Pseudorandom function $F$
- Pseudorandom generator $G: S \rightarrow X$

Idea: shared key is $F$ applied to published values

- $F$ itself kept secret
- Publish program that computes $F$,
  - but only if user supplies proof that they are allowed to
Our Construction (w/ Trusted Setup)

How to establish shared group key?

1. \( x_1 = G(s_1) \)
2. \( x_2 = G(s_2) \)
3. \( x_3 = G(s_3) \)
4. \( x_4 = G(s_4) \)

\( s_1 \leftarrow S \)

\( s_2 \)

\( s_3 \)

\( s_4 \)
Our Construction (w/ Trusted Setup)

\[
F(P(y_1, \ldots, y_n, s, i)) \{
    \text{If } G(s) \neq y_i, \text{ output } \bot \\
    \text{Otherwise, output } F(y_1, \ldots, y_n)
\}
\]
Our Construction (w/ Trusted Setup)

\[ K_{ABCD} = P'(x_1, x_2, x_3, x_4, s_1, 1) \]
Security of Our Construction

Adversary sees $P'$ and the $X_i$, wants to learn $F(x_1, ..., x_n)$

$F \{ y_1, ..., y_n, s, i \} \{
    \text{If } G(s) \neq y_i, \text{ output } \perp
    \text{Otherwise, output } F(y_1, ..., y_n)
\}$
Step 1: Replace $x_i$

Draw $x_i$ uniformly at random

- Security of $G$: adversary cannot tell difference

$$F( y_1, ..., y_n, s, i ) \{$$
  If $G(s) \neq y_i$, output $\bot$
  Otherwise, output $F(y_1, ..., y_n)$
$$\}$$

Observation: if $X$ is much larger than $S$, all $x_i$ are outside range of $G$, w.h.p.
Punctured PRFs [BW’13, KPTZ’13, BGI’13, SW’13]

Can give out code to evaluate $F$ at all but a single point $z$

Security: given $F^z$, $t=F(z)$ indistinguishable from random

$F^z \approx F^z \rightarrow t = F(z)$

$t \leftarrow T$
Step 2: Puncture F

Let \( z = (x_1, ..., x_n) \)
Puncture F at z, and abort if input is z

\[
P_z( y_1, ..., y_n, s, i ) \{
    \text{If } G(s) \neq y_i, \text{ output } \bot
    \text{If } (y_1, ..., y_n) = z, \text{ output } \bot
    \text{Otherwise, output } F^z(y_1, ..., y_n)
\}
\]

Inputs where P₂ differs from P?
- Only \((x_1,...,x_n,s,i)\) where \(G(s) = x_i\)
- W.h.p. no such input exists
- iO: P₂ indistinguishable from P
Step 3: Simulate

Simulate view of adversary, given $F^z$

$$F^z$$

$$P_2( y_1, ..., y_n, s, i ) \{$$
- If $G(s) \neq y_i$, output $\bot$
- If $(y_1, ..., y_n) = z$, output $\bot$
- Otherwise, output $F^z(y_1, ..., y_n)$

$$\text{Security of } F: k = F(z) \text{ indist. from a random key}$$
Removing Trusted Setup

As described, our scheme needs trusted setup

Observation: Obfuscated program can be generated independently of publishing step

\[
\begin{align*}
\text{P}( y_1, \ldots, y_n, s, i ) & \{ \\
& \text{If } G(s) \neq y_i, \text{ output } \bot \\
& \text{Otherwise, output } F(y_1, \ldots, y_n) \\
\}
\end{align*}
\]

Untrusted setup: user 1 generates \( P' \), sends with \( x_1 \)
Multiparty Key Exchange Without Trusted Setup

\[ x_1 = G(s_1) \]
\[ x_2 = G(s_2) \]
\[ x_3 = G(s_3) \]
\[ x_4 = G(s_4) \]
Broadcast Encryption
Broadcast Encryption

\[ P' \]

\[ x_D \]
\[ x_1 \]
\[ x_2 \]
\[ x_3 \]
\[ x_4 \]

\[ x_1 = G(s_1) \]
\[ x_2 = G(s_2) \]
\[ x_3 = G(s_3) \]
\[ x_4 = G(s_4) \]

dummy user

S_1
S_2
S_3
S_4
Broadcast Encryption

- Replace unintended recipients with dummy
- Compute shared key for protocol
  - Ex: $k = F(x_1, x_D, x_D, x_4)$
- Use shared key to encrypt message
Broadcast Encryption

Private key scheme: empty ciphertext header

Public broadcast key scheme: a single $x_i$ value

Additional Properties:
• Distributed – users and broadcaster each generate their own parameters
• Can be used with existing RSA keys (under plausible assumptions)
Other Constructions

Recipient private broadcast encryption
- Ciphertext size: $\lambda + n$
- Secret key size: $\lambda$
- Public key size: $\text{poly}(n, \lambda)$

Traitor tracing
- Ciphertext size: $\lambda + \log(n)$
- Secret key size: $\lambda$
- Public key size: $\text{poly}(\log(n), \lambda)$
Open Questions

Reduce public key sizes
• Using *differing-inputs* obfuscation [ABGSZ’13]
• From iO?

Other primitives from iO
• FHE?

Thanks!